NASA Technical Memorandum 4149

The Aerospace Energy Systems Laboratory: A BITBUS Networking Application

Richard D. Glover and Nora O'Neill-Rood

NOVEMBER 1989

NASA

NASA Technical Memorandum 4149

The Aerospace Energy Systems Laboratory: A BITBUS Networking Application

Richard D. Glover and Nora O'Neill-Rood
Ames Research Center
Dryden Flight Research Facility
Edwards, California

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1989

THE AEROSPACE ENERGY SYSTEMS LABORATORY: A BITBUS NETWORKING APPLICATION

Richard D. Glover* and Nora O'Neill-Rood†

NASA Ames Research Center
Dryden Flight Research Facility
Edwards, California

Abstract

The NASA Ames-Dryden Flight Research Facility has developed a computerized aircraft battery servicing facility called the Aerospace Energy Systems Laboratory (AESL). This system employs distributed processing with communications provided by a 2.4-megabit BITBUS local area network (LAN). Customized handlers provide real-time status, remote command, and file transfer protocols between a central system running the iRMX-II operating system and ten slave stations running the iRMX-I operating system. This paper describes the hardware configuration and software components required to implement this BITBUS application.

Nomenclature

AESL	Aerospace Energy Systems Laboratory
Ames-Dryden	NASA Ames Research Center, Dryden Flight Research Facility
BIOS	basic input/output system
BSC	battery station controller
BSL	battery systems laboratory
EIOS	extended input/output system
EPROM	erasable PROM
IDD	initial data descriptor
IEEE	Institute of Electrical and Electronic Engineers
I/O	input/output
iDCM	distributed control microcontroller
iDCX	distributed control executive
iRMX	real-time multitasking executive
Kbyte	kilobyte
LAN	local area network
LSBX	Linear Systems Ltd. SBX (clock/calendar module)
Mbit	megabit
Mbyte	megabyte
PROM	programmable read-only memory
RAC	remote access and control
RAM	random-access memory
SBC	single-board computer
SBX	single-board expansion
SCP	status and control panel

*Aerospace Engineer.
†Systems Engineer

Introduction

The NASA Ames Research Center's Dryden Flight Research Facility (Ames-Dryden) has undertaken a modernization effort to upgrade its existing battery systems laboratory (BSL). The BSL is a computerized facility employed in the servicing of a variety of large storage batteries used in the research aircraft operating at Ames-Dryden. The second-generation BSL will be called the Aerospace Energy Systems Laboratory (AESL) (Glover and O'Neill-Rood, 1988) and will become operational in late 1990. As shown in figure 1, the AESL will be a distributed system consisting of a central system linked to a number of battery stations by a local area network (LAN).

A photograph of the AESL engineering prototype battery station bench is shown in figure 2. The battery being serviced is connected to the charger and is monitored by the monitor plate clamped to its top. The controller subsystem is a Multibus I system which inputs battery current, voltage, and temperature, and outputs control over the charger through the power control subsystem. In addition, the controller provides operator input/output (I/O) interfaces for the status and control panel (SCP) and the barcode gun. Because of the harsh environment, the battery stations have no rotating drives whatsoever; instead two 128-kilobyte

(Kbyte) magnetic bubble memory cassettes provide local storage for program software and data.

As shown in figure 3 the AESL central system consists of two desktop chassis housings plus a terminal and line printer. The main chassis contains a Multibus I cardcage plus peripherals consisting of a floppy drive, a tape drive, and a magnetic bubble memory cassette interface. The auxiliary chassis contains two 51-megabyte (Mbyte) hard drives, one for software production files and the other for an integrated database containing battery servicing configuration files and data records.

This report will discuss first how the requirements for the AESL led to the choice of BITBUS as LAN, and the configuration of the hardware components required to implement it. Secondly, the report will describe the evolution of a NASA-developed protocol used to satisfy the AESL communications requirements. Finally, the report will outline the software elements which comprise this protocol and will give examples of typical BITBUS operations.

Requirements

The base line AESL physical requirements are for ten battery stations to be installed in a servicing area adjacent to the operations room containing the central system. A daisy chained serial bus was selected to minimize the amount of wiring; for this reason a star configuration (either serial or parallel) was not considered. The total bus length required was 100 ft and a master–slave protocol was used to reduce the cost of the station controllers as much as possible.

The AESL system functional requirement involving the greatest amount of LAN traffic is the exchange of files between the central system integrated database and the bubble memories in the battery stations. During battery servicing, a battery station may ask for the download of a specification file related to a particular battery so that it may properly configure the controller for a servicing run. The central system, on the other hand, must periodically upload data records generated during battery servicing so that they may be added to the database archives for access by interactive jobs. These files are typically 250–550 bytes in length and during periods of high activity may reach a rate of 50 file transfers an hour.

A second functional requirement is the real-time monitoring of battery station activity from an interactive job. This involves passing blocks of status information from the selected station to the job running in the central system. The blocks run 500–600 bytes in length, but since they are only generated on demand, the total traffic volume is relatively small.

The third functional requirement is for command capability to allow a battery technician to exercise remote control over operations at a selected battery station. This need may arise since the AESL is engineered to operate unattended, including overnight and weekend operations. The AESL multiuser interface will allow authorized personnel to dial into the system over telephone lines to both monitor operations and exercise control if necessary. This capability would be used infrequently and the effect on bus loading is expected to be negligible.

A final set of functional requirements involves housekeeping operations such as station polling, time synchronization, and station bubble memory garbage collection. Station polling must occur round-robin in such a way that no station waits more than 1 sec for servicing. Based on ten stations, the polling rate would be 10 Hz. Time synchronization involves passing a 32-bit date–time quantity from the central system to a station when requested (once a minute). Ten stations would generate ten such request–response message pairs a minute and constitutes perhaps 1 percent of bus loading. Station bubble memory garbage collection involves purging stale files after they have been uploaded to the central system. Such purge operations would normally be done by the system manager only during slack periods and thus would not adversely affect bus loading.

Hardware Configuration

Early in the AESL design effort it was decided that a LAN mechanized with BITBUS hardware and software components would meet all of the requirements previously discussed, and would be highly cost effective as well. The SBX344A module was chosen as the interfacing hardware module since it is specifically designed to provide a BITBUS gateway to any single board computer (SBC) having a single board extension (SBX) connector. As shown in figure 4, the SBC used in the AESL central system is the 286/12, while the SBC used in each of the ten battery station controllers (BSC) is the 86/35. Both board types have SBX connectors, and an SBX344A module will be installed on all 11 boards. The bus itself will consist of two runs of twisted shielded pair cable 100 ft in length, one pair for data and the other for the clock.

The configuration of the 11 SBX344A modules varies slightly depending on which SBC is host. All modules must be configured for 2.4-Mbit/sec synchronous operation and have socket U14 configured to receive a D2764A erasable programmable read-only memory (EPROM). The 286/12 is the master node and will be located at one end of the BITBUS. Its SBX344A module has been given decimal address 28 and is configured with terminating resistors on both the clock and data pairs. The remaining SBX344A modules will be configured for addresses one through ten, and will not have terminating resistors installed. External terminating resistors are used at the end of the BITBUS farthest from the master node.

The SBX344A module as received from the factory has its 8044 microcontroller chip programmed with distributed control microcontroller (iDCM)-44 release 2.0 firmware. This firmware is a preconfigured version of the distributed control executive (iDCX)-51 release 2.0 which provides a single resident task 0 called the remote access and control (RAC) task. This firmware provides all the commands nec-

essary for operation of the AESL LAN, and therefore no additional iDCX-51 software is required.

From the viewpoint of the iDCM-44 firmware, the SBC board which hosts the SBX344A module is considered an "extension" which means the firmware communicates with the SBC through its parallel port (the firmware communicates with the BITBUS through its serial port). Most communications are from the master node's extension (the SBC286/12) to one of the slave node extensions (an SBC86/35). However, an extension can communicate with its own node using address 0FFH or with a remote node (rather than the remote node's extension) by setting the destination extension bit to 0 in the message header. When communication with a particular slave node's extension is broken, the 286/12 communicates with that slave node directly (RAC command RESET_STATION) and with its own node to declare the slave node inactive (RAC command OFFLINE). The SBC86/35 board at a slave node communicates with its own node during initialization in order to read its address jumpers (using RAC command READ_IO at address 0FFH) to identify its own particular station number.

The iDCM-44 firmware provides a default BITBUS message length of 20 bytes. Since each message has a 7-byte header, this leaves 13 bytes (or 65 percent) of the message available for data. When BITBUS was first considered for the AESL, it was decided that a longer message was mandatory to improve the LAN performance when passing large blocks of data. Fortunately, the release 2.0 version of the iDCM-44 firmware has a feature (shown in fig. 5) called an initial data descriptor (IDD) which is invoked at SBX344A power-up. An IDD permits changing the default values for clock period, clock priority, and message length (RQSYSBUFSIZE), and also permits reserving a portion of memory so that it cannot be used to buffer messages. The firmware checks address 1000H to see if either programmable read-only memory (PROM) or random access memory (RAM) has been installed in socket U14. If it finds the check pattern 0AA55H, it chains any initial data (and task) descriptors into the initialization sequence.

All AESL SBX344A modules have a D2764A EPROM installed in socket U14 which contains the IDD shown in figure 5. The total available memory for iDCM-44 release 2.0 is 108 bytes. On the advice of Intel Corporation this IDD was structured to redefine the message length to 54 bytes, thus allowing two messages to be buffered. This change has lowered the message overhead from 35 percent to 13 percent and has improved bus bandwidth considerably. The clock period and priority are left unchanged and no memory is reserved.

Bus Protocol Evolution

In the early stages of AESL software development (Glover, 1988b), the protocol shown in figure 6 was tested extensively. The destination task field of the BITBUS message header was used to specify 1 of 16 possible functions to be performed within the BSC. This protocol has two major disadvantages: (1) it is highly application specific and therefore inflexible and (2) it does not permit concurrent communications with more than one central system job. For these reasons, a new approach was considered necessary even though the protocol functioned well and provided valuable experience in the uploading and downloading of large segments of data using BITBUS message packets.

The second phase of protocol development took the approach of structuring all bus traffic around the management of real-time multitasking executive (iRMX) objects in the slave extensions. The central system could create and delete mailboxes, could send and receive message segments at any desired mailbox, and could perform four operations on segments: create, delete, upload, and download. Application-specific aspects of communications under this protocol are hidden within the segments being exchanged and concurrent processing is assured by each job having its own set of objects. Figure 7 shows the steps involved in a master–slave exchange under this protocol. The weakness of this protocol is that the master must either have *a priori* knowledge of the proper destination mailbox when initiating an exchange, or there must be agreed-upon conventions (such as use of null tokens) for accessing specific mailboxes created by the slave.

The final phase of protocol development began with the ground rule that each slave would create two mailboxes which could be accessed by the master using certain destination task identifiers in the message header. One would be called the command mailbox, to which jobs in the master would send command segments (forcing some sort of action by the slave). The other would be known as the request mailbox, which the master would periodically poll to see if a requestor task within the slave had sent a request segment (asking the master to take some action). This protocol, shown in figure 8, eliminates the need for the master to create and delete mailboxes, but essentially retains the other six operations of the purely iRMX object based approach previously mentioned. For both approaches, concurrent processing ends after the command segment is sent to the command mailbox if there is only one task performing command servicing. Similarly, slave requestor tasks must wait in turn if there is only one request servicing task doing the polling in the master.

Battery Station Software Modules

The AESL battery stations employ SBC86/35 boards running a partial RMX I release 7 configuration consisting of nucleus, basic input/output system (BIOS), and a single user job as shown in figure 9. No device drivers are needed for either the SBX344A or the Linear Systems Ltd. clock–calendar module (LSBX) since only a single task within the user job interfaces to either device.

The tasks involved with BITBUS operations are shown in figure 9 and of those, only the BITBUS task is application independent. The BITBUS task creates the command

mailbox and the request mailbox. The command servicing task waits for command message segments posted by the BITBUS task and if closed loop, generates a reply message. There are two requestor tasks in the AESL application which can send request segments to the request mailbox: one which occasionally requests files to be downloaded from the central system, and another which once a minute requests a date–time download for clock synchronization. This automatic synchronization is an important feature of any distributed system because it permits a single master clock to pass along all time changes to the slaves, as well as correcting normal drift. In the AESL, synchronization is especially important because file names for data files are generated from the date–time at the instant of the snapshot.

When the BITBUS task is created, the address jumpers on the SBX344A board are read and that address becomes the station number for that extension processor. Files sent to the central system are tagged with this station number so the source can be identified. The BITBUS task is configured as a polling task which checks the SBX344A module every 20 ms to determine if any messages have been received, and if so, generates and sends immediate reply messages. If no traffic is received for 10 sec, a flag is set which inhibits requestor tasks from sending requests to the request mailbox.

Central System Software Modules

The AESL central system is a 286/12 system 310 with 5 Mbytes of zero wait state RAM running a fully configured iRMX II.3 operating system. Multichannel communication boards allow the coexistence of several interactive jobs using round-robin scheduling and 50-ms windows. As shown in figure 10, there are two NASA developed device drivers linked to the BIOS and two I/O jobs created by the extended I/O (EIOS). The clock device driver and the clock I/O job relate to a battery powered clock–calendar module used for master timekeeping, and are not involved with the BITBUS LAN in any way.

The BITBUS device driver was designed to recognize all BIOS functions except F$READ, F$WRITE, and F$SEEK. The function F$ATTACH$DEV causes the SBX344A module to be reset, while F$DETACH$DEV, F$OPEN, and F$CLOSE simply return E$OK. The function F$SPECIAL accepts only a subfunction code of 8001H and interprets it to mean a BITBUS message transaction where the auxiliary pointer points to the BITBUS message to be sent:

```
declare   bitbus$msg    structure  (
            link          word,
            length        byte,
            flags         byte,
            node          byte,
            task$ids      byte,
            cmd$resp      byte,
            dat (248)     byte  );
```

The BITBUS device driver would be invoked as follows:

```
call rq$s$special  (  bitbus$connection,
                      8001H,
                      @bitbus$msg,
                      nil,
                      @exception  );
```

The response message is written on top of bitbus$msg so it is imperative that the data field be long enough for the longest possible response.

The polling I/O job is a high priority job which first reads the contents of the file :CONFIG:STATIONS and builds a list of stations to be polled. Thereafter it iterates at 10 Hz, polling the set of all stations thought to be active and for each such scan polls one station from those thought to be off-line. Each poll causes the status of the station to be updated depending on whether or not a reply was received. If a reply was received and a request for service was fetched, the request is immediately processed and the response is sent back.

The interactive jobs which involve sending commands to a station are linked to a set of routines which provide interleaving of bus traffic using RQSSPECIAL calls as previously shown. A standard interface to these routines has been developed which employs a structure of the following form:

```
declare   command$msg    structure  (
            node           byte,
            function       byte,
            count          word,
            exception      word,
            actual         word,
            buffer$ptr     pointer,
            string$ptr     pointer  );
```

The top-level command routine is invoked as follows:

```
call   command$io   ( @command$msg  );
```

This routine is necessarily application-dependent but in turn it is linked to a number of supporting routines which are solely protocol-dependent. For a given application, all interactive jobs would be bound to the same command routine (and supporting routines) which would interpret the function code and perform the proper BITBUS I/O operation.

Bus Operations

As stated earlier, bus operations can be initiated at one of two points: a central system job which commands a particular station to take some action, or a station task which sends a request segment to its request mailbox. All testing to date has been performed with a single station: the

engineering prototype battery bench (which was given address 27). Stress tests have been performed where three interactive jobs have simultaneously commanded the station to upload a large status segment. Some slight delay is noticed occasionally in the response time of the interactive job's display, but these delays are generally less than 2 sec. File transfers of 250 to 550 bytes have been tested extensively and delays are generally 2 to 3 sec. A large part of this delay is caused by the slow response time of the magnetic bubble cassettes in the BSC.

Figure 11 shows the elements involved in a typical closed-loop command sequence. The circled numbers in figure 11 correspond to the following steps:

1. The interactive I/O job calls the command routine with a function code involving closed-loop command,
2. A command segment is created and its token is returned,
3. The command segment is filled using multiple download transmissions,
4. The command segment is sent as a closed-loop command, a reply mailbox is created, and its token is returned,
5. The command routine inserts an appropriate time delay for the reply segment to be generated and sent. The command segment is deleted by the slave's command servicing task after the reply segment has been generated,
6. The command routine begins to solicit the reply segment token and repeats until received. The reply mailbox is deleted by the slave's BITBUS polling task after the reply segment token has been successfully picked up,
7. The reply segment is emptied using multiple upload transmissions,
8. The reply segment is deleted,
9. The information is passed back to the interactive job.

To enhance the performance of the protocol, command segments are downloaded and reply segments are uploaded only if the information will not fit into the available free space in the BITBUS message. The amount of free space in the message varies from 39 to 45 bytes depending on the message type (fig. 8). When the information will fit into this free space, the presence of immediate data is signalled by the command or reply segment token being set equal to the value selector$of(nil). Thus whenever a message is received which involves a segment, the token must be checked and immediate data used when appropriate. In the case of command information, the slave's BITBUS polling task will create the command segment when necessary to receive the immediate data. In the case of reply information, the BITBUS polling task will send the information back as immediate data whenever possible and delete the reply segment. Otherwise, the recipient is responsible for deleting the reply segment after it has been uploaded.

Figure 12 shows the logic employed in the polling I/O job. As described earlier, the job fetches the file :CONFIG:STATIONS during initialization and builds a polling list. It also creates a status array which indicates whether a station is believed to be "asleep" or "awake", and initializes it to all asleep. All awake stations (if any) are polled once and then the next station from those classified asleep is polled. One station is polled every 100 ms and its status is updated; this polling rate is a compromise between servicing delay and central system processor loading. If a reply was received and a request for service was fetched, the request is immediately processed and the response is sent back. For a STATIONS file consisting of

1,2,3,4,5,6,7,8,9,10,27

the polling rate for the single active station 27 is approximately 85 times a minute. This rate could be expected to drop to 60 a minute with ten stations active.

Concluding Remarks

In late 1990 the NASA Ames-Dryden AESL will become operational as a semi-automated facility for the servicing of aircraft batteries. This facility will be a distributed Multibus I system with a central system running iRMX II and ten operator positions running iRMX I linked by a BITBUS network. A prototype network, consisting of the master node and a single slave node, is now operational and has proven highly reliable at 2.4 Mbit/sec over 100 ft of bus. The BITBUS message length has been increased to 54 bytes by taking advantage of the IDD feature in the latest release of the BITBUS DCX. A NASA developed protocol permits a mix of interactive and I/O jobs executing within the central system to communicate concurrently with any of the servicing positions. This protocol makes possible BITBUS applications beyond distributed control to intersystem networking, including file transfers and other communications involving large segments of data.

References

Glover, Richard D., "Aerospace Energy Systems Laboratory: Requirements and Design Approach," *proceedings of 34th International Instrumentation Symposium*, Albuquerque, NM, pp. 359–363, May 2–6, 1988.

Glover, Richard D., and O'Neill-Rood, Nora, "The Aerospace Energy Systems Laboratory: Hardware and Software Implementation," *proceedings of the Fifth International iRMX User's Group Conference*, Schaumburg, IL, pp. 219–237, Nov. 14–15, 1988.

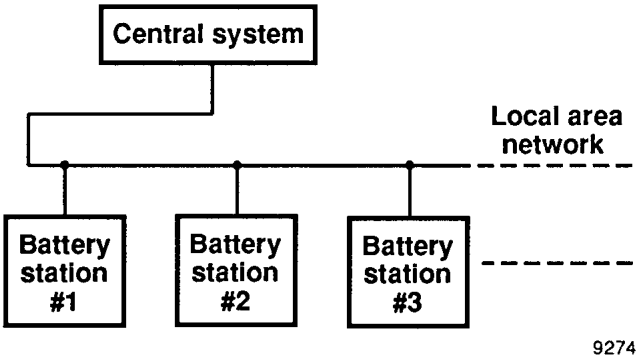

Figure 1. AESL overview.

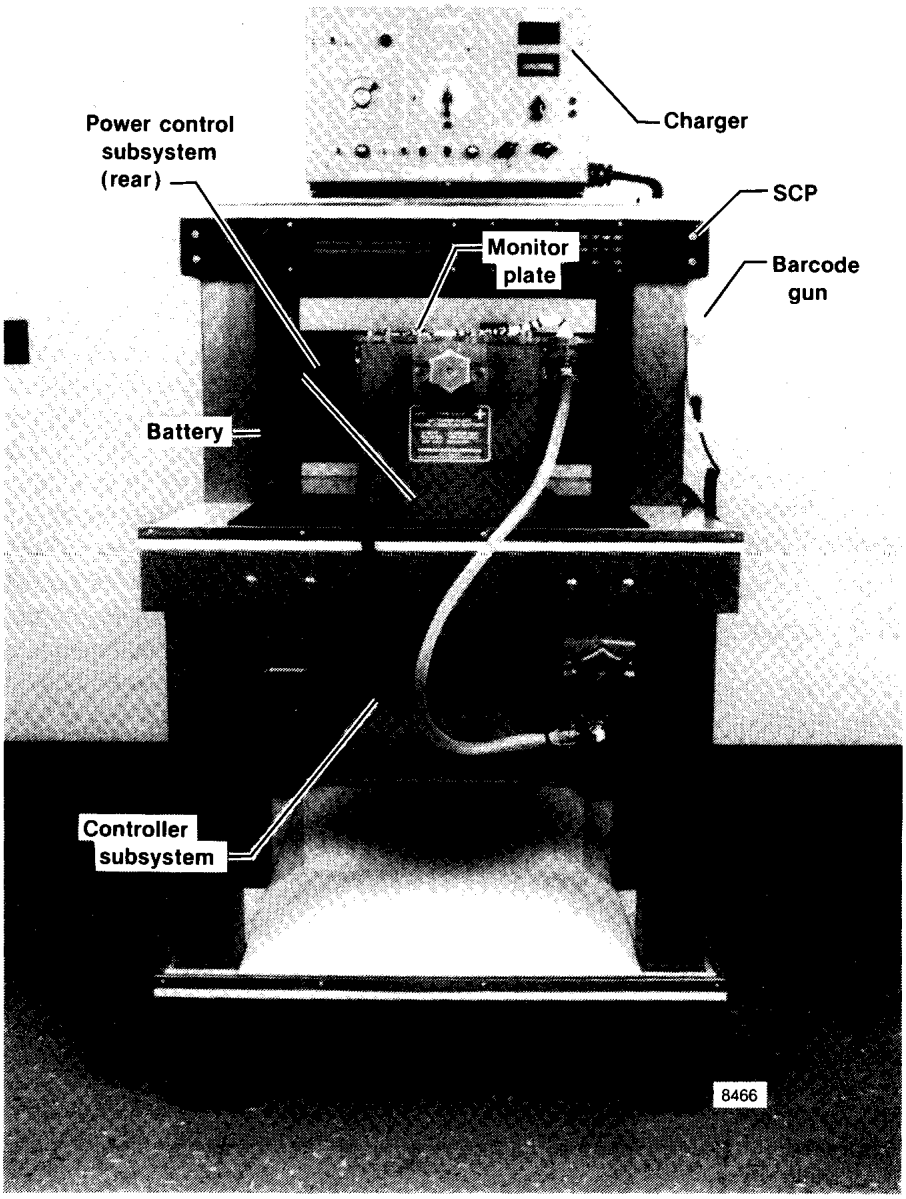

Figure 2. Battery station bench.

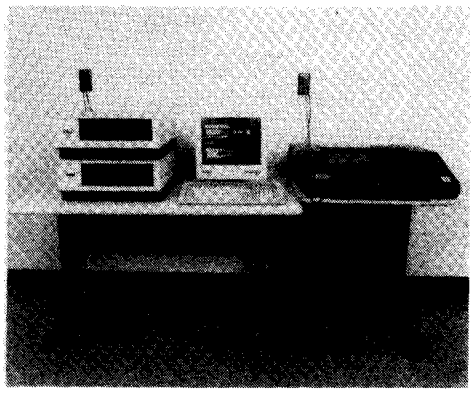

Figure 3. Central system console.

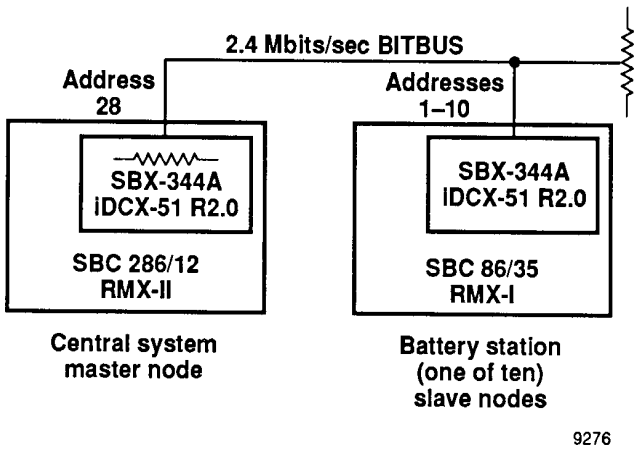

Figure 4. BITBUS hardware configuration.

Address	Contents	Description
1000H	DW 0AA55H	; check pattern
1002H	DW 0	; block type (IDD)
1004H	DW 0	; RQCLOCKTICK (no change)
1006H	DB 0	; RQCLOCKPRIORITY (no change)
1007H	DB 54	; RQSYSBUFSIZE (new value)
1008H	DB 0	; reserved memory (none)
1009H	DW 0	; next block (none)

Figure 5. Initial data descriptor.

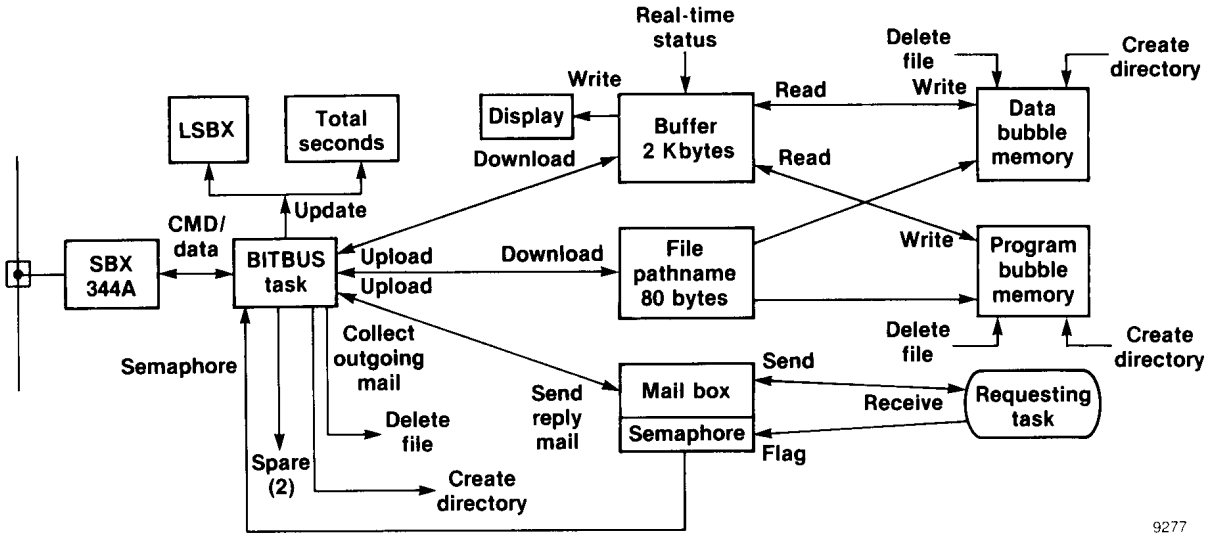

Figure 6. Initial protocol design.

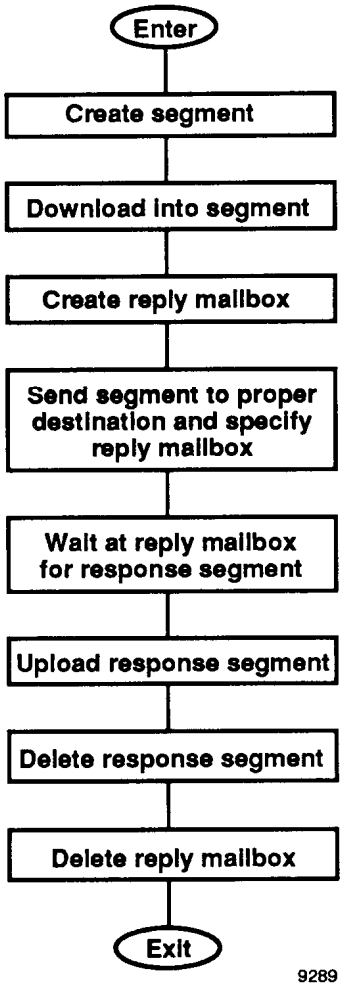

Figure 7. iRMX object based protocol.

Task	Function	Send to slave extension	Receive from slave extension
1	Issue open-loop command	`command$segment$token` `immediate$data(45)byte*`	`status`
2	Issue closed-loop command	`command$segment$token` `immediate$data(45)byte*`	`status` `reply$mailbox$token`

3	Retrieve closed-loop command reply	`reply$mailbox$token`	`status`
			`reply$segment$token`
			`reply$segment$size`
			`immediate$data(41)byte*`

4	Check service request mailbox	– nothing –	`status`
			`request$segment$token`
			`request$segment$size`
			`service$mailbox$token`
			`immediate$data(39)byte*`

5	Deliver service response	`service$mailbox$token`
		`service$segment$token`
		`immediate$data(43)byte*`
			`status`

6	Create segment	`segment$size`	`status` `segment$token`
7	Delete segment	`segment$token`	`status`

8	Upload from segment	`offset`
		`segment$token`
		`[command$response=#]`
			`status`
			`data$bytes(#)byte`

9	Download into segment	`offset`
		`segment$token`
		`data$bytes(#)byte`
		`[command$response=#]`
			`status`

*If immediate data appended, `segment$token = selector$of(nil)`

Figure 8. Final protocol design.

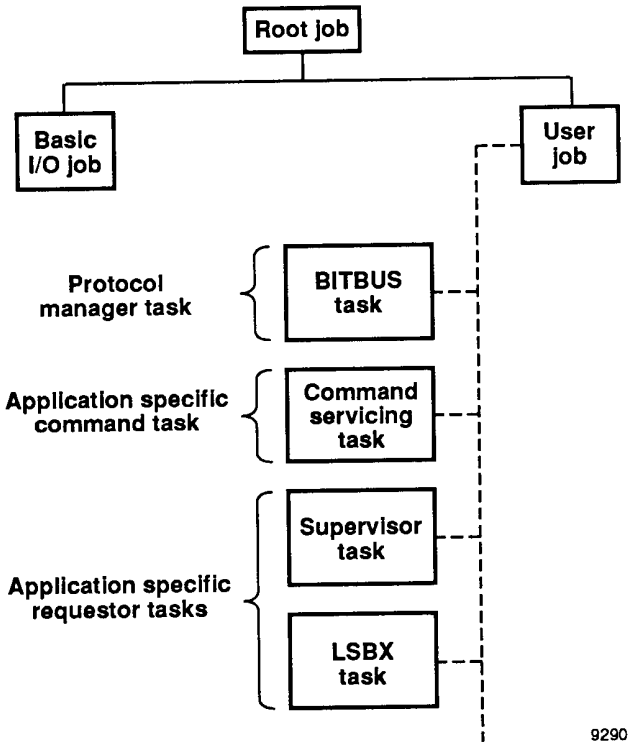

Figure 9. Battery station software.

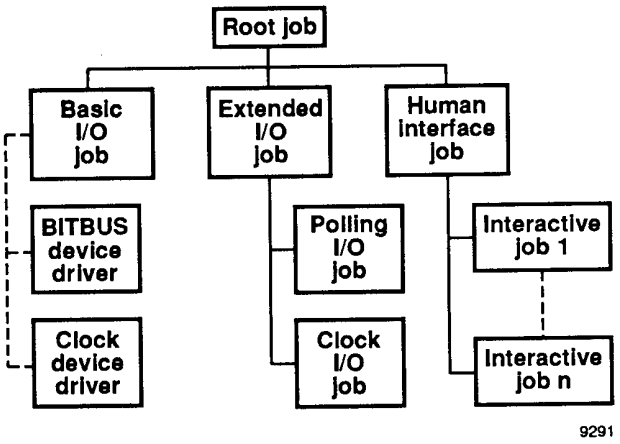

Figure 10. Central system software.

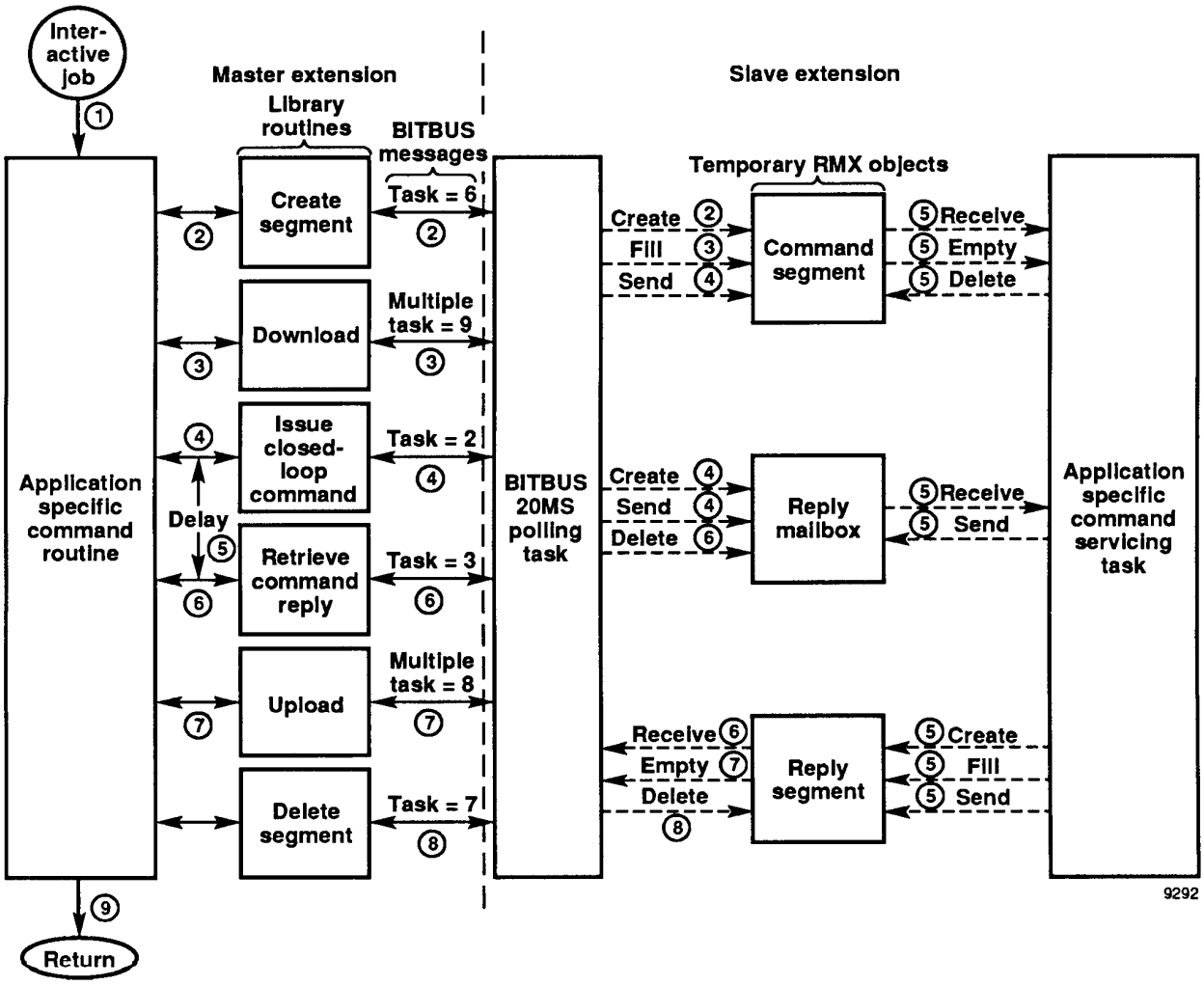

Figure 11. Typical closed-loop command sequence.

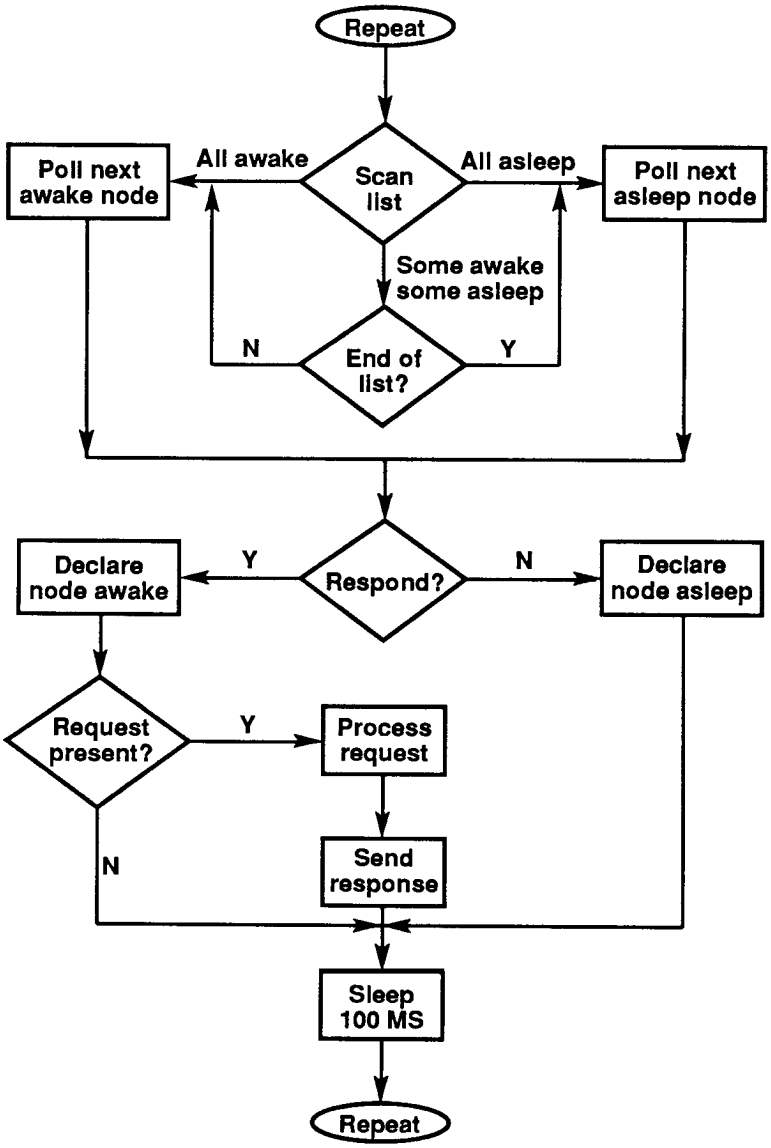

Figure 12. Polling I/O job.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-4149		

4. Title and Subtitle	5. Report Date
The Aerospace Energy Systems Laboratory: A BITBUS Networking Application	November 1989
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Richard D. Glover and Nora O'Neill-Rood	H-1569
	10. Work Unit No.
	RTOP 992-23-05
9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Ames Research Center Dryden Flight Research Facility P.O. Box 273, Edwards, CA 93523-5000	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

Prepared as conference paper for presentation at iRUG 6th International Conference, Bethesda, Maryland, November 13–14, 1989.

16. Abstract

The NASA Ames-Dryden Flight Research Facility has developed a computerized aircraft battery servicing facility called the Aerospace Energy Systems Laboratory (AESL). This system employs distributed processing with communications provided by a 2.4-megabit BITBUS local area network (LAN). Customized handlers provide real-time status, remote command, and file transfer protocols between a central system running the iRMX-II operating system and ten slave stations running the iRMX-I operating system. This paper describes the hardware configuration and software components required to implement this BITBUS application.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
BITBUS; Distributed processing; iRMX; Master–slave; Multibus; Multiuser	Unclassified — Unlimited Subject category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	15	A02

NASA FORM 1626 OCT 86

**For sale by the National Technical Information Service, Springfield, VA 22161-2171.*

NASA-Langley, 1989